FastStream

Airt - Пастухов Никита

PiterPy 2024

Брокеры и Python

в 2024 году

airt

План на сегодня

- Эволюция инструментов разработчика
- Требования к тулингу в эру **FastAPI**
- История одного велосипеда
- Плюсы и минусы **FastStream**

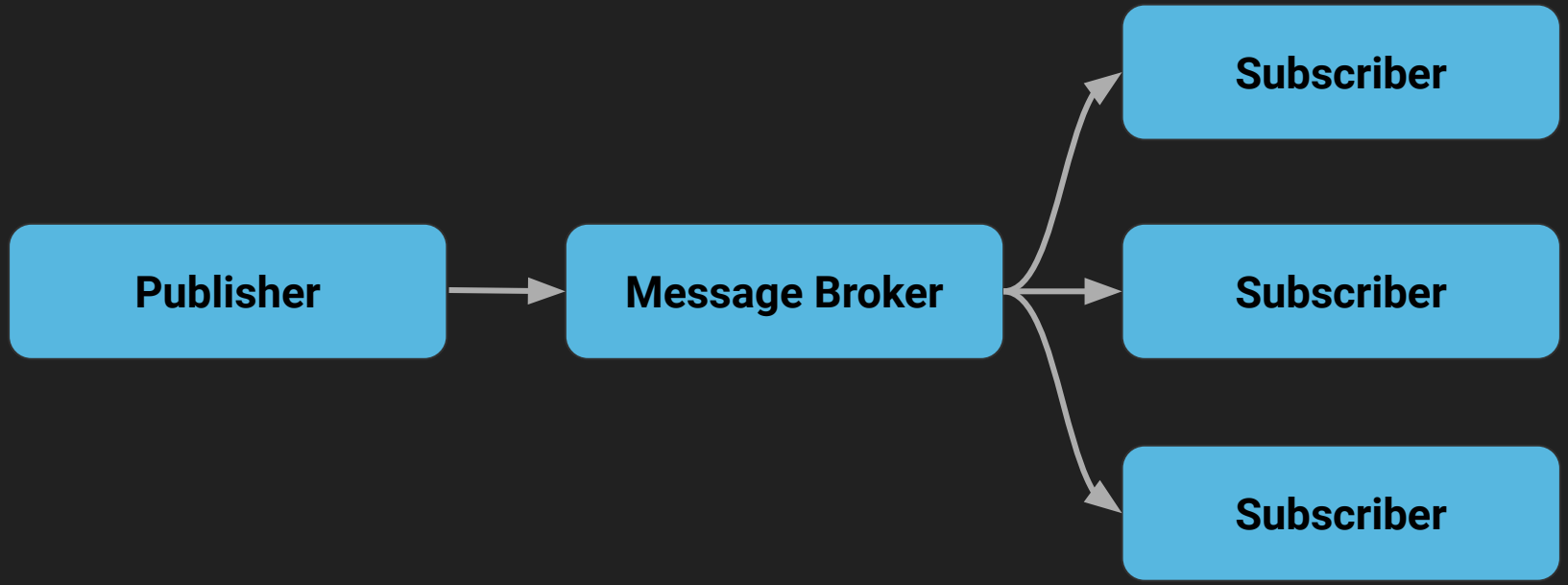
airt

Пастухов Никита

- Разработчик - разрабатываю (6 лет)
- Опыт с асинхронностью - **RabbitMQ** (4 года)
- Стало больно - создал **Propan**
- Сотрудничаю с **airt**, разрабатываю **FastStream**

Асинхронные сервисы

airt



MQ / Streaming tools

airt

FastStream – это фреймворк для

- ~~RabbitMQ~~
- ~~Kafka~~
- ~~Nats~~
- ~~Redis~~
- асинхронных сервисов

FastStream

```python
from faststream import FastStream
from faststream.rabbit import RabbitBroker

broker = RabbitBroker()
app = FastStream(broker)

@broker.subscriber("input-queue")
async def handler(msg):
    ...  # обработка
```

Эволюция инструментов разработчика

airt

На примере

airt

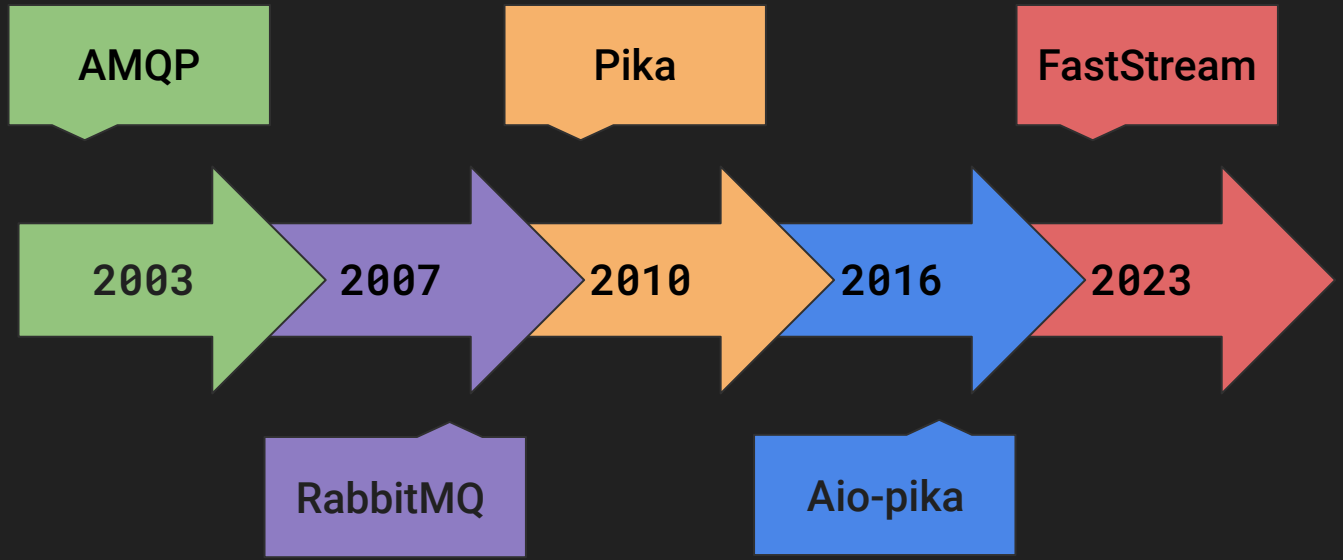
Дмитрий Орлов

Автор **aio-pika** / **aiormq**

https://github.com/mosquito

https://youtu.be/EFVwilQvv_4?si=tO2aYEkVc73Ru9cx

Немного истории



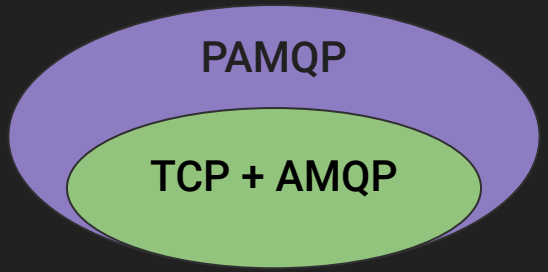
Уровни абстракций

Уровни абстракций

airt

https://github.com/gmr/pamqp

51 ★

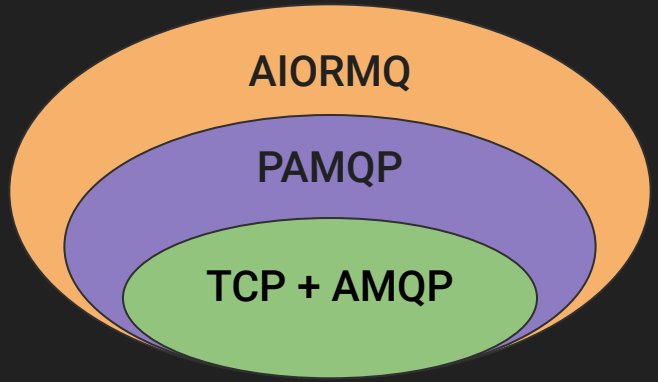


Уровни абстракций

airt

https://github.com/mosquito/aiormq

264 ★

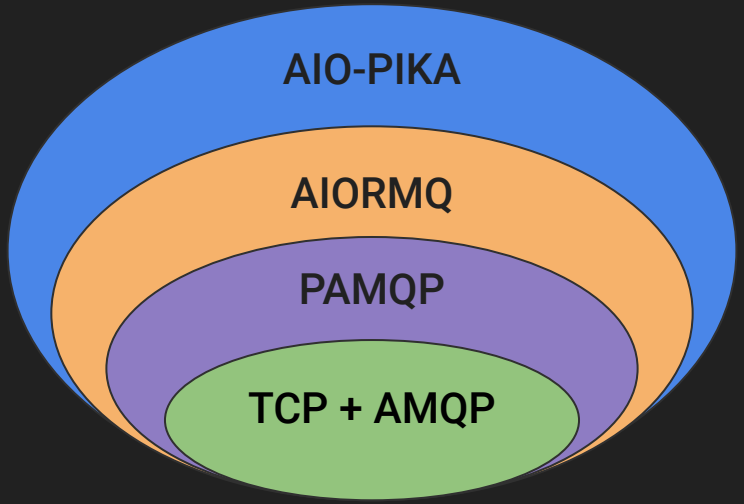


Уровни абстракций

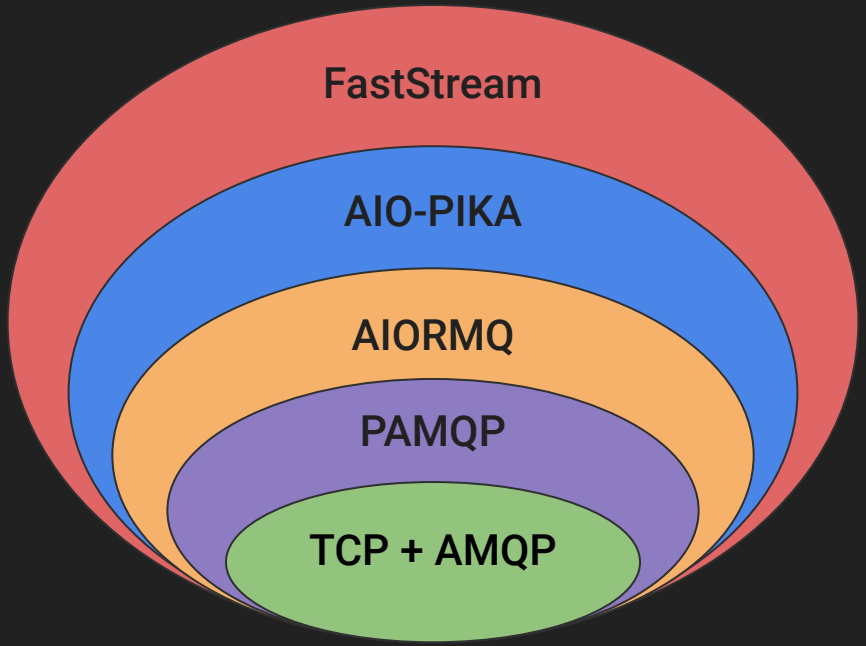
airt

https://github.com/mosquito/aio-pika

1.2k ★



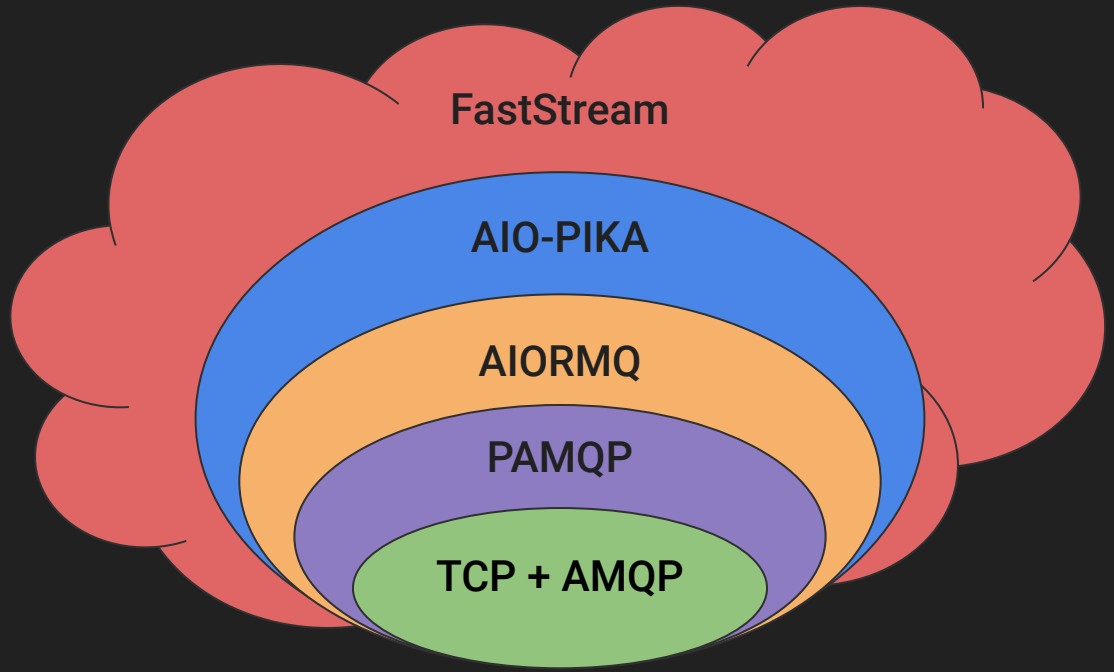
Уровни абстракций



https://github.com/airtai/faststream

2.4k ★

Уровни абстракций

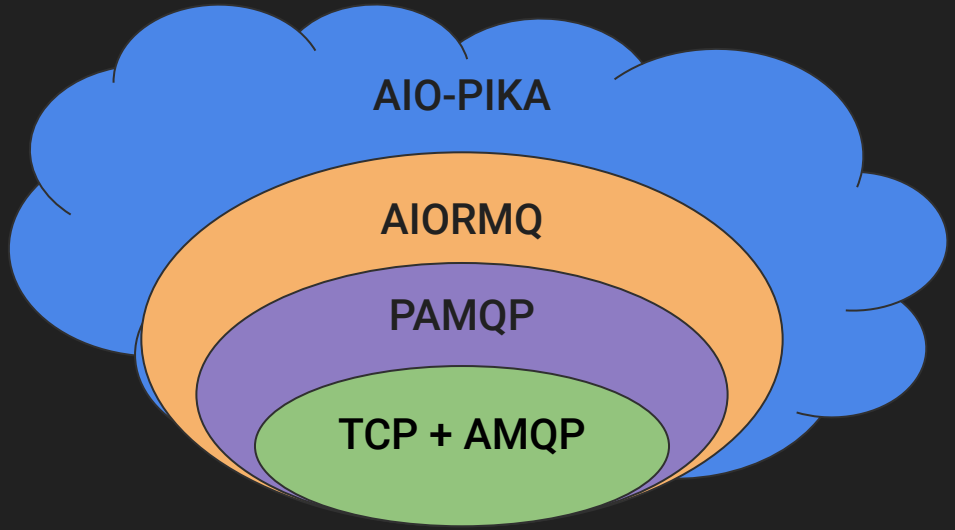


FastStream

- сериализация
- роутеры
- DI
- мидлвари
- CLI

Caxap?

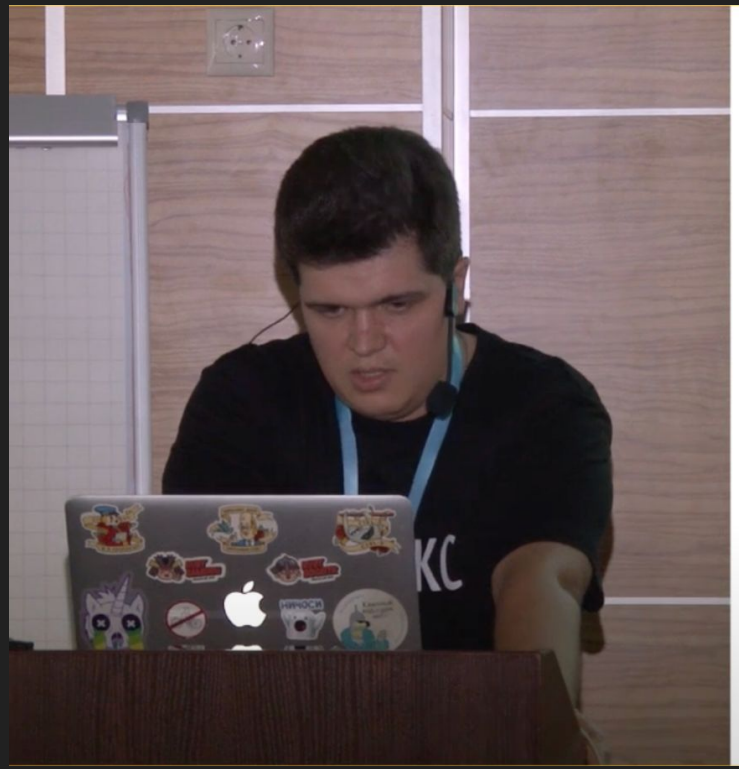
airt



aiormq

- channel
- publish
- consume

Сахар?



API

```
〉class Connection
〉class Channel
〉class Queue
〉class Exchange
〉class Transaction
〉class Message
```

все поля и заголовки сообщения в одном месте

```
〉class IncomingMessage(Message)
```

+ методы **ack()/nack()/reject()**

```
〉class RobustConnection(Connection)
```

airt

Сахар, который позволяет провалиться в детальки

Еще сахар?

Зачем шаблоны?

- Не нужно писать каждый раз одно и тоже
- Либо сложный AMQP 0.9 или применить уже реализованный и оттестированный шаблон
- Код легче поддерживать: фокус на бизнес логике а не на реализации коммуникации с брокером

30

Еще сахар?

RPC caller

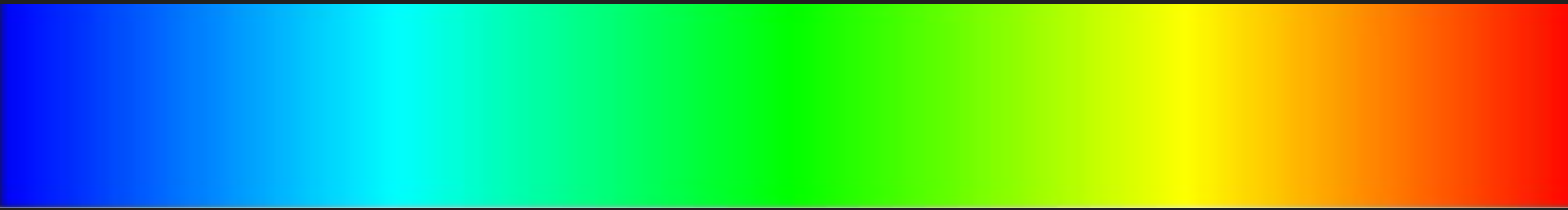
```python
import asyncio
import aio_pika
from aio_pika.patterns import RPC


file_urls = [
    ('png', 'http://storage.local/logo.png'),
    ...
]


async def main():
    connection = await aio_pika.connect_robust()

    async with connection:
        async with connection.channel() as channel:
            rpc = RPC(channel)

            results = await asyncio.wait([
                rpc.call(f'process_{kind}', kwargs={'url': url})
                for kind, url in file_urls
            ])

            print(results)
```

airt

Пишем бизнес-логику, а не инфраструктуру

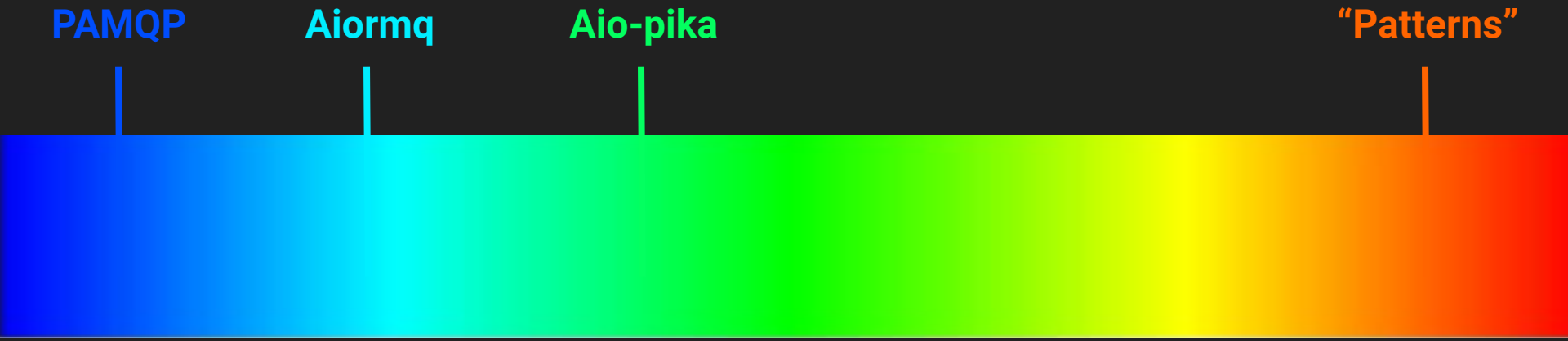
Градация сахара

Sugar-free **Сахарная кома**



Градация сахара

airt



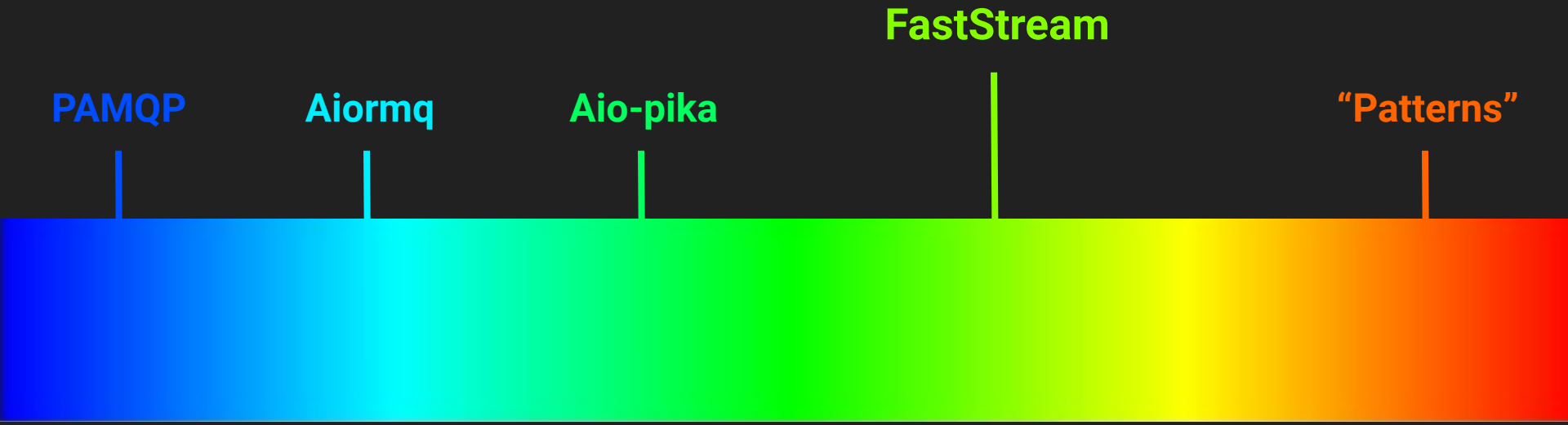
Другие “паттерны”

airt

- **Celery**
- **Faust**
- **Bytewax**
- **Quick-streams**
- **Celery-like** (dramatiq, taskiq, arq, saq, etc)

Где тут FastStream?

airt



Идеология FastStream

airt

- **Прозрачность** – юзер всегда понимает, что делает фреймворк
- **Модульность** – меняй что хочешь и как хочешь
- **Батарейки** – не на все случаи жизни
- **Ограниченный контекст** – только клиент для брокера, ничего более

airt

```python
import aio_pika

async def main():
    connection = await aio_pika.connect_robust(
        "amqp://guest:guest@127.0.0.1/"
    )

    async def connection:
        channel = await connection.channel()

        queue = await channel.declare_queue(
            "test_queue"
        )

        async with queue.iterator() as queue_iter:
            async for message in queue_iter:
                async with message.process():
                    ...  # обработка
```

aio-pika

airt

```python
import aio_pika

async def main():
    connection = await aio_pika.connect_robust(
        "amqp://guest:guest@127.0.0.1/"
    )

    async def connection:
        channel = await connection.channel()

        queue = await channel.declare_queue(
            "test_queue"
        )

        async with queue.iterator() as queue_iter:
            async for message in queue_iter:
                async with message.process():
                    ...  # обработка
```

- императивный синтаксис
- тестирование
- нет документации на сервис
- observability

Тулинг HTTP

airt

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def index():
    return "Hello, world!"
```

Тулинг HTTP

airt

- Удобный **API**
- **OpenAPI + SwaggerUI**
- Удобство тестирования
- **Observability**

Что делаем?

airt

```python
from fastapi import FastAPI

app = FastAPI() Broker()

@app.post("/") subscriber("input")
async def handler(data: InputModel) -> Prediction:
    prediction = model.predict(data)
    return prediction
```

PROPAN

FastKafka

airt

FastStream = PROPAN + FastKafka

FastStream

```python
from faststream import FastStream
from faststream.rabbit import RabbitBroker

broker = RabbitBroker()
app = FastStream(broker)

@broker.subscriber("input-queue")
async def handler(msg):
    ...  # обработка
```

airt

aio-pika

```python
import aio_pika

async def main():
    connection = await aio_pika.connect_robust(
        "amqp://guest:guest@127.0.0.1/"
    )

    async def connection:
        channel = await connection.channel()

        queue = await channel.declare_queue(
            "test_queue"
        )

        async with queue.iterator() as queue_iter:
            async for message in queue_iter:
                async with message.process():
                    ...  # обработка
```

Императивный код

airt

```python
import aio_pika

async def main():
    connection = await aio_pika.connect_robust(
        "amqp://guest:guest@127.0.0.1/"
    )

    async def connection:
        channel = await connection.channel()

        queue = await channel.declare_queue(
            "test_queue"
        )

        async with queue.iterator() as queue_iter:
            async for message in queue_iter:
                async with message.process():
                    ...  # обработка
```

Законнектись

Объяви очередь

Перебирай сообщения

Декларативный код

airt

```python
from faststream import FastStream
from faststream.rabbit import RabbitBroker

broker = RabbitBroker()
app = FastStream(broker)

@broker.subscriber("input-queue")
async def handler(msg):
    ...  # обработка
```

Хочу подписчика

Еще FastStream

airt

```python
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker()
app = FastStream(broker)

@broker.subscriber("input-subject")
async def handler(msg):
    ...  # обработка
```

```python
from faststream import FastStream
from faststream.kafka import KafkaBroker

broker = KafkaBroker()
app = FastStream(broker)

@broker.subscriber("input-topic")
async def handler(msg):
    ...  # обработка
```

```python
from faststream import FastStream
from faststream.redis import RedisBroker

broker = RedisBroker()
app = FastStream(broker)

@broker.subscriber("input-channel")
async def handler(msg):
    ...  # обработка
```

FastStream

airt

```python
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker()
app = FastStream(broker)

@broker.subscriber("input-subject")
async def handler(data: InputData):
    prediction = model.predict(data)
```

FastStream

```python
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker()
app = FastStream(broker)

@broker.subscriber("input-subject")
async def handler(data: InputData):
    prediction = model.predict(data)
```

Входящее сообщение

```python
from pydantic import BaseModel

class InputData(BaseModel):
    username: str
    user_id: int
```

Ожидаем JSON

```json
{
    "username": "Nikita",
    "user_id": 1
}
```

Публикация

airt

Поддерживает

- Python-примитивы
- типы **std**
- **pydantic.BaseModel**

```python
async with NatsBroker() as broker:
    await broker.publish(
        message={"username": "Nikita", "user_id": 1},
        subject="input-subject",
    )
```

Результат

airt

```python
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker()
app = FastStream(broker)

@broker.subscriber("input-subject")
async def handler(data: InputData) -> Prediction:
    prediction = model.predict(data)
    return prediction
```

Куда?

RPC over MQ

airt

```python
async with NatsBroker() as broker:
    response = await broker.request(
        message=InputData(...),
        subject="input-subject",
    )
```

Синхронный запрос *

* не для всех брокеров

Reply To

Ждем результат

```python
@broker.subscriber("reply")
async def handle_prediction(msg: Prediction):
    ...


async with NatsBroker() as broker:
    await broker.publish(
        message=InputData(...),
        subject="input-subject",
        reply_to="reply",
    )
```

Куда отправить результат

Publisher

airt

```python
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker()
app = FastStream(broker)

@broker.subscriber("input-subject")
@broker.publisher("out-subject")
async def handler(data: InputData) -> Prediction:
    prediction = model.predict(data)
    return prediction
```

Куда отправить результат

Services chain

airt

```python
@broker.subscriber("input-subject")
@broker.publisher("out-subject")
async def service_1(data: InputData) -> Prediction:
    prediction = model.predict(data)
    return prediction


@broker.subscriber("out-subject")
@broker.publisher("final")
async def service_2(data: Prediction) -> Prediction:
    return data


@broker.subscriber("final")
async def service_3(data: Prediction) -> None:
    ...
```

А как запустить?

airt

```python
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker()
app = FastStream(broker)
```

Что ты такое?*

* Объект для **CLI**

```
faststream run main:app --workers 2

faststrean run main:app --reload
```

Не очень умный

Тестирование

airt

- тестирование **бизнес-логики** (забить на контракты)
- тестирование с реальным брокером (“удобно” в **CI**)
- мокаем транспорт (в каждом сервисе)

Тестирование

- ~~тестирование **бизнес-логики** (забить на контракты)~~
- ~~тестирование с реальным брокером (“удобно” в **CI**)~~
- ~~мокаем транспорт (в каждом сервисе)~~
- **Берем FastStream**

Тестирование с FastStream

airt

Исходная функция

```python
@broker.subscriber("input-subject")
async def handler(username: str, user_id: int):
    if user_id < 1:
        raise ValueError()
    return f"User {username} created!"
```

Тестирование с FastStream

airt

Тестирование входящего контракта

```python
import pytest
from faststream.nats import TestNatsBroker

@pytest.mark.asyncio()
async def test_incorrect():
    async with TestNatsBroker(broker) as test_broker:
        with pytest.raises(ValueError):
            await test_broker.publish(
                {"username": "Nikita", "user_id": 0},
                subject="input-subject",
            )
```

Тестирование с FastStream

airt

Тестирование исходящего контракта

```python
import pytest
from faststream.nats import TestNatsBroker

@pytest.mark.asyncio()
async def test_correct():
    async with TestNatsBroker(broker) as test_broker:
        result = await test_broker.request(
            {"username": "Nikita", "user_id": 1},
            subject="input-subject",
        )

        assert result.body == b"User Nikita created!"
```

E2E Тестирование

airt

```python
async with TestNatsBroker(broker, with_real=True) as test_broker:
    await test_broker.publish(...)
```

Observability

- Логирование
- Healthcheck’и
- Трейсинг
- Метрики? (уже вот-вот)

Логирование

```python
from faststream import FastStream, Logger
from faststream.nats import NatsBroker

broker = NatsBroker()
app = FastStream(broker)

@broker.subscriber("in")
async def handler(msg, logger: Logger):
    logger.info(msg)
```

```
1970-01-01 00:00:17,137 INFO     - FastStream app starting...
1970-01-01 00:00:17,142 INFO     - in   |            - `Handler` waiting for messages
1970-01-01 00:00:17,142 INFO     - FastStream app started successfully! To exit, press CTRL+C
1970-01-01 00:00:17,143 INFO     - in   | 0fd3e176-f - Received
1970-01-01 00:00:17,143 INFO     - in   | 0fd3e176-f - Hello, FastStream!
1970-01-01 00:00:17,143 INFO     - in   | 0fd3e176-f - Processed
```

Healthcheck’и

```python
from faststream.nats import NatsBroker
from faststream.asgi import AsgiFastStream, make_ping_asgi

broker = NatsBroker()

app = AsgiFastStream(
    broker,
    asgi_routes=[
        ("/health", make_ping_asgi(broker, timeout=5.0))
    ]
)
```

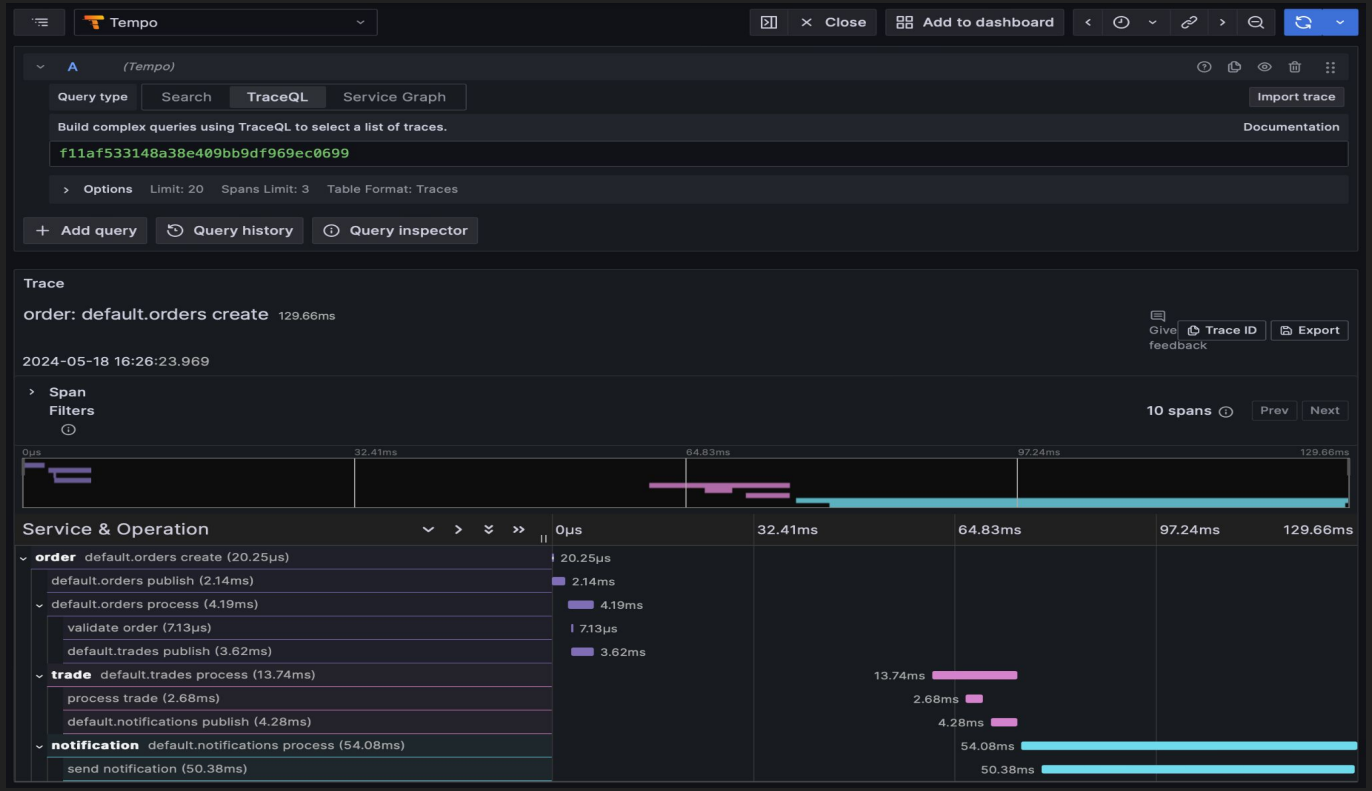
HTTP?

Запуск через ASGI-сервер

```
uvicorn main:app
```

Трейсинг

airt



OpenTelemetry boilerplate

airt

```python
from opentelemetry import trace
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import OTLPSpanExporter
from opentelemetry.sdk.resources import Resource
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor

resource = Resource.create(attributes={"service.name": "faststream" })
tracer_provider = TracerProvider(resource=resource)
tracer_provider.add_span_processor(
   BatchSpanProcessor(OTLPSpanExporter(endpoint="http://127.0.0.1:4317"))
)
trace.set_tracer_provider(tracer_provider)
```

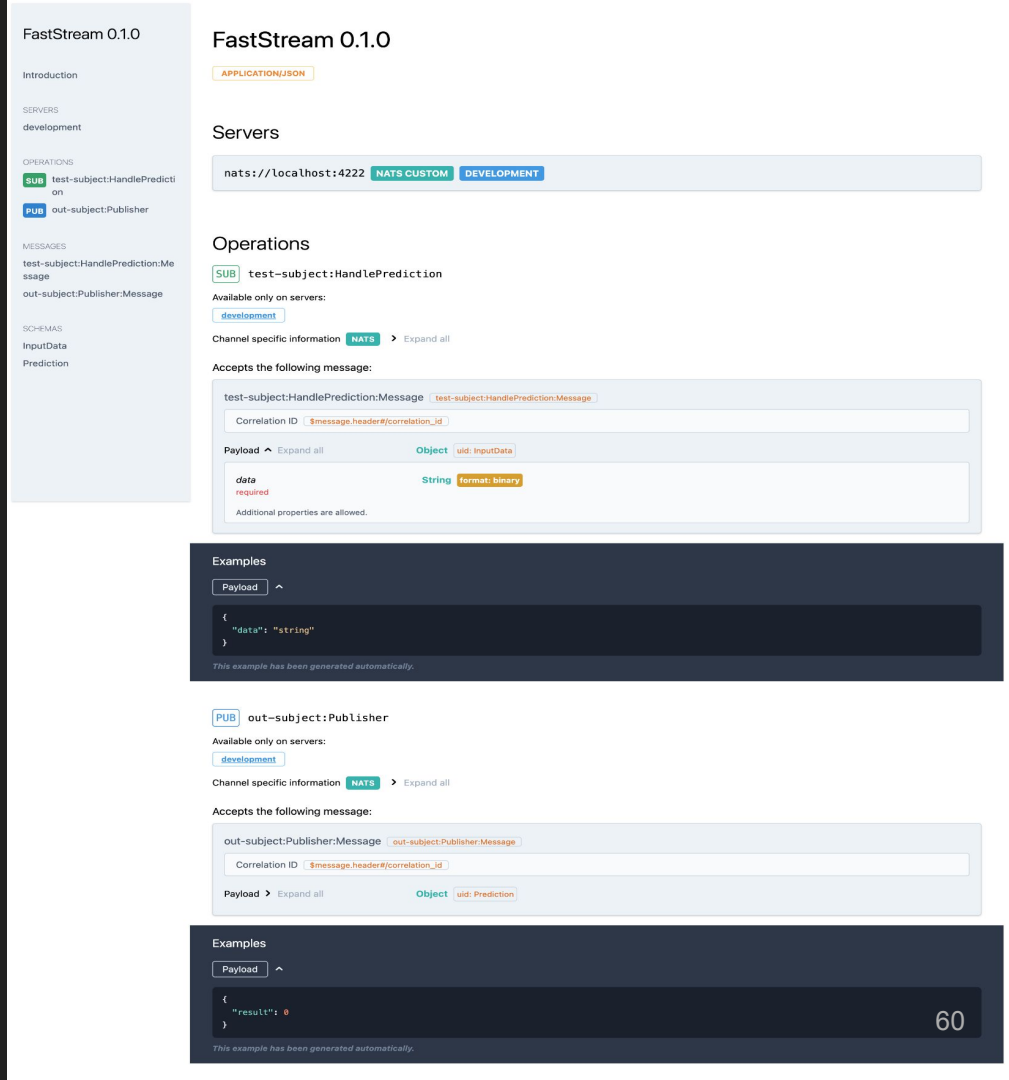
OpenTelemetry Middleware

```python
from faststream.nats import NatsBroker
from faststream.nats.opentelemetry import NatsTelemetryMiddleware

broker = NatsBroker(
    middlewares=(
        NatsTelemetryMiddleware(tracer_provider=tracer_provider),
    ),
)
```

AsyncAPI

https://www.asyncapi.com/en

```python
from pydantic import BaseModel
from faststream import FastStream
from faststream.nats import NatsBroker

broker = NatsBroker()
app = FastStream(broker)

class InputData(BaseModel):
    data: bytes

class Prediction(BaseModel):
    result: float

@broker.subscriber("test-subject")
@broker.publisher("out-subject")
async def handle_prediction(
    msg: InputData
) -> Prediction:
    predict = model.predict(msg)
    return predict
```



Другие фичи

airt

- **Depends** - как в FastAPI
- **Context** - похоже на нормальный DI
- **Router**'ы, **Middleware** и прочий фарш
- Интеграция с **FastAPI** (и чем угодно)

airt

Критика FastStream

Глобальный Broker

airt

Не нравится

```python
from faststream.nats import NatsBroker

broker = NatsBroker("nats://localhost:4222")

@broker.publisher("next-step")
@broker.subscriber("user-created")
async def handler(username: str, user_id: int):
    return f"Hello, {username}!"
```

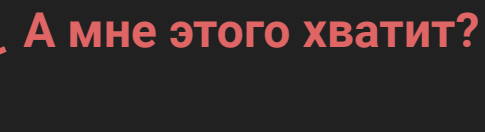
НЕ глобальный Broker

airt

```python
from faststream.nats import NatsBroker, NatsRouter

router = NatsRouter()

@router.publisher("next-step")
@router.subscriber("user-created")
async def handler(username: str, user_id: int):
    return f"Hello, {username}!"

def create_broker():
    broker = NatsBroker("nats://localhost:4222")
    broker.include_router(router)
```

Теряем фичи брокера

```python
from faststream.nats import NatsBroker

broker = NatsBroker()

@broker.subscriber("user-created")
async def handler(username: str, user_id: int):
    return f"Hello, {username}!"
```



НЕ теряем фичи брокера

airt

```python
from faststream.rabbit import RabbitBroker, RabbitExchange, ExchangeType

broker = RabbitBroker()

@broker.subscriber(
    "test",
    exchange=RabbitExchange("test-exchange", type=ExchangeType.TOPIC),
)
async def handler():
    ...
```

НЕ теряем фичи брокера

```python
from faststream import FastStream, Logger
from faststream.nats import NatsBroker

broker = NatsBroker()
app = FastStream(broker)

@broker.subscriber("some-key", kv_watch="bucket")
async def handler(msg: int, logger: Logger):
    logger.info(msg)

@app.after_startup
async def test():
    kv = await broker.key_value("bucket")
    await kv.put("some-key", b"1")
```

“Прибитый” pydantic

airt

```python
from faststream.nats import NatsBroker

broker = NatsBroker(validate=False)
```

“Выключаем” pydantic

Кастомная сериализация

```python
from faststream import BaseMiddleware
from faststream.rabbit import RabbitBroker, RabbitMessage

class MsgPackMiddleware(BaseMiddleware):
    async def publish_scope(self, call_next, msg, **options):
        return await call_next(
            msgpack.dumps(msg, use_bin_type=True),
            **options,
        )

def decode_message(msg: RabbitMessage):
    return msgpack.loads(msg.body)

broker = RabbitBroker(
    decoder=decode_message,
    middlewares=(MsgPackMiddleware,),
)
```

Пакуем исходящие сообщения

Распаковываем входящие

Зачем вам FastStream

airt

- **TTM** - пишем бизнес-логику, а не инфраструктуру
- Требуется **меньше экспертизы** в брокерах
- **Качество кода** - не собираем грабли сами
- **Качество системы** - мониторинг, тестируемость и документация

Планы на будущее

airt

- Prometheus
- БОЛЬШЕ БРОКЕРОВ (SQS, MQTT, ZMQ, ...)
- Несколько брокеров в одном приложении
- Поддержка AsyncAPI 3.0 (0.6.0)
- Расширение комьюнити, привлечение контрибуторов

Star, comment and subscriber!

- GitHub
- Discord EN
- Telegram RU
- Cookiecutter template
- Мой другой доклад о **FastStream**
- Спасибо команде: Tvrtko, Davor, Kumaran, Harrish, Robert and Hajdi